

1 METHODS AND APPARATUS FOR MODIFYING
2 PROGRAMS STORED IN READ ONLY MEMORY
3

4 BACKGROUND OF THE INVENTION
5

6 1. Field of the Invention

7 The invention relates to embedded systems. More
8 particularly, the invention relates to methods and apparatus for
9 modifying embedded system programs stored in read only memory
10 (ROM).
11

12 2. State of the Art

13 Digital computing systems are typically known to include
14 hardware and software. Software is typically stored on a writable
15 medium such as a magnetic disk. As most computer users know,
16 software often contains errors or mistakes which prevent it from
17 functioning properly. Software vendors often publish updates or
18 "patches" to correct errors in software. An update is typically a
19 new complete version of the original software which is intended to
20 replace the entire original program. The original software is
21 erased from the writable medium and the replacement software is
22 written to the writable medium in place of the original. A patch
23 is a program which is run to modify the original software. The
24 patch program finds the portion(s) of the software which need to
25 be replaced and overwrites them with replacement code.

1 Today, digital computing systems are ubiquitous and often
2 unseen. These systems are designed to perform specialized
3 functions such as controlling the operation of an appliance, a
4 motor vehicle, or a computer peripheral device such as a modem or
5 a printer. Such computing systems are usually referred to as
6 "embedded systems". They include a microprocessor and software
7 which is typically stored on a read only memory (ROM). ROM is
8 advantageous because it is non-volatile, small, inexpensive, and
9 energy efficient. Program code stored on ROM is usually called
10 "firmware" rather than software because once it is stored on ROM,
11 the code cannot be modified. The code takes on the quality of
12 hardware in the sense that in order to change it, the physical ROM
13 device must be replaced. Indeed, many embedded systems have
14 "socketed" ROM so that the ROM can be easily unplugged and
15 replaced with a new ROM if the program needs to be changed.
16 However, that is not always practical or convenient.

17
18 One known method for alteration of firmware programs in ROM-
19 based systems is disclosed in U.S. Patent Number 4,607,332, issued
20 on August 19, 1986 to Goldberg. The method allows for dynamic
21 alteration of ROM programs with the aid of random access memory
22 (RAM) and through the use of the standard linkages associated with
23 subroutine calls in the system processor. When each ROM-based
24 routine is written, one program statement is a call to a ROM-
25 located processing routine which searches a RAM-located data

1 structure. If there exists a correspondence between information
2 passed on the call to the processing routine and certain elements
3 of the data structure, a RAM-based program is substituted for the
4 ROM-based program. After adjusting the processor to the states
5 just after the call to the ROM-based program, the processing
6 routine effects a transfer to the replacement routine in RAM. The
7 location of this replacement routine is also found in the data
8 structure. The main disadvantage of the method proposed by
9 Goldberg is that it is inefficient. It uses up too much space in
10 ROM and RAM. It also requires an external compiler/interpreter.
11 Since the processing function is ROM-based, it is limited in scope
12 and potential to be modified. The ROM-based function also
13 presents excessive processing overhead since it must perform a
14 search for possible replacement code even when no replacement code
15 exists

16
17 Another system for altering ROM-based firmware is disclosed
18 in U.S. Patent Number 5,740,351, issued on April 14, 1998 to
19 Kasten. The system relies on an "extensible interpreter", i.e. a
20 modified FORTH kernel and a plurality of customizable call outs
21 (CCOs). CCOs are present in the ROM-based program. When a CCO is
22 encountered during execution of the program, the modified FORTH
23 kernel takes control and looks for the called function or
24 parameter. If it is found (in RAM or in ROM), it is executed or
25 fetched and the result is returned by the modified FORTH kernel to

1 the next instruction in the ROM program. The system is primarily
2 intended for interactive use with a dumb terminal during
3 "debugging" of the ROM-based program. CCOs in RAM must be defined
4 using the modified FORTH kernel. The main advantage of Kasten
5 over Goldberg is that Kasten does not require an external
6 compiler/interpreter. Disadvantages of Kasten are that it is
7 limited to modifications made using the FORTH programming language
8 and it is inefficient, requiring that a substantial part of ROM be
9 devoted to the modified FORTH kernel.

10
11 Still another system for altering software in embedded
12 systems is disclosed in U.S. Patent Number 5,901,225, issued on
13 May 4, 1999 to Ireton et al. The system includes an embedded
14 system device coupled to an external memory. The device includes
15 a non-alterable memory, including firmware, coupled to a
16 processor. The device further includes a relatively small amount
17 of patch RAM within the device also coupled to the processor.
18 Patches are loaded from the external memory into the patch RAM.
19 The device further includes a means for determining if one or more
20 patches are to be applied. If the device detects a patch to be
21 applied, the system loads the patch from the external memory into
22 the patch RAM. The device also includes a breakpoint register.
23 When the value of the program counter of the processor equals the
24 value in the breakpoint register, a patch insertion occurs, i.e.,
25 the processor deviates from executing firmware to executing patch

1 instructions. The system described by Ireton et al. is quite
2 complex.

3
4 SUMMARY OF THE INVENTION
5

6 It is therefore an object of the invention to provide methods
7 and apparatus for modifying firmware code stored in a read only
8 memory.

9
10 It is also an object of the invention to provide methods and
11 apparatus for modifying firmware code stored in a read only memory
12 which make efficient use of RAM.

13
14 It is another object of the invention to provide methods and
15 apparatus for modifying firmware code stored in a read only memory
16 which are relatively simple in architecture.

17
18 It is also an object of the invention to provide methods and
19 apparatus for modifying firmware code stored in a read only memory
20 which do not require any decision making elements in ROM to
21 support future code modifications.

22
23 It is another object of the invention to provide methods and
24 apparatus for modifying firmware code stored in a read only memory

1 which do not require any processor specific or processor dependent
2 elements.

3
4 It is still another object of the invention to provide
5 methods and apparatus for modifying firmware code stored in a read
6 only memory which are applicable to any programming language.

7
8 It is also an object of the invention to provide methods and
9 apparatus for modifying firmware code stored in a read only memory
10 which do not limit the scope of future code modifications.

11
12 It is another object of the invention to provide methods and
13 apparatus for modifying firmware code stored in a read only memory
14 which incur minimum processing overhead.

15
16 It is still another object of the invention to provide
17 methods and apparatus for modifying firmware code stored in a read
18 only memory which minimize the size of ROM code segments needed to
19 be replaced to fix an error.

20
21 In accord with these objects which will be discussed in
22 detail below, the apparatus of the invention includes an embedded
23 device having program code stored in ROM and an on-board or
24 external RAM for storing modified code segments. The methods of
25 the invention include structuring the ROM-based firmware so that a

1 RAM-based function is called prior to each potentially modifiable
2 code segment. Prior to modifying the firmware, a dummy function
3 is stored in RAM so that every call to RAM is simply returned to
4 ROM. When a segment of code is to be modified, a replacement is
5 stored in RAM and indexed by the return address of the function
6 call. The system of the present invention is efficient as it uses
7 very little RAM. It does not require any ROM-based decision
8 making elements; and it is not limited to a particular programming
9 language or processor. The system of the invention is most
10 suitable for use in a computer peripheral which communicates with
11 a higher level controller, e.g. a personal computer, from which
12 replacement code can be downloaded. Alternatively, replacement
13 code in RAM can be loaded by a small bootstrap program (i.e. a
14 run-time system initialization program) stored in replaceable
15 external ROM.

16
17 Additional objects and advantages of the invention will
18 become apparent to those skilled in the art upon reference to the
19 detailed description taken in conjunction with the provided
20 figures.

21 22 BRIEF DESCRIPTION OF THE DRAWINGS

23
24 Figure 1 is a simplified block diagram of an embedded system
25 according to the invention;

Figure 2 is a schematic illustration of the organization of ROM-based code according to the invention;

Figure 3 is a schematic illustration of the dummy function in RAM according to the invention;

Figure 4 is a schematic illustration of the replacement code in RAM indexed to return address; and

Figure 5 is a simplified flowchart of the operation of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to Figure 1, an embedded system 10 according to the invention includes a processor 12 which is coupled to a firmware ROM 14 and a RAM 16. As illustrated in Figure 1, the RAM 16 is ideally much smaller than the ROM 14, for example approximately one one hundredth the size of the ROM.

Turning now to Figure 2, according to the invention, the firmware stored in the ROM 14 is divided into code segments, e.g. 18a-18d, 20a-20d, 22a-22d, and 24a-24d. Prior to the start of each code segment, a call function with a (calling and) return

1 address is placed, e.g. at 18, 20, 22, and 24. The locations of
2 the call functions are preferably organized for the most efficient
3 use of RAM. It might seem practical to locate each call function
4 at each logical break in the code, e.g. at the start of each new
5 routine. However, in the case of very long routines, it can be
6 more efficient to place call functions equally spaced throughout
7 the routine. This will be better understood after considering the
8 code listings below. Essentially, whenever replacement code is
9 used, all of the code from the call function until the corrected
10 code is reached is replaced. Thus if there are one thousand lines
11 of code between call functions and only the six hundredth line of
12 that code needs to be changed, six hundred lines will be replaced
13 nonetheless. If the call functions were spaced every one hundred
14 lines throughout the thousand line routine, no more than one
15 hundred lines would need to be replaced. Thus, a balance must be
16 struck between the number of call functions placed throughout the
17 ROM-based code and the amount of RAM needed to make code
18 replacements.

19
20 Figure 3 illustrates the structure of RAM 16 when none of the
21 code segments in ROM is to be replaced. A dummy function 26,
22 including instruction lines 26a-26d resides in RAM 16. Whenever a
23 call function (18, 20, 22, 24 in Figure 2) is executed, the dummy
24 function 26 responds by returning program execution to the return

1 address (which is usually the next line after the calling function).

2
3 Figure 4 illustrates the structure of RAM 16 when some of the
4 code segments in ROM are to be replaced. In this case, the Return
5 in the Dummy Function is replaced with a branch to the
6 corresponding Patch Center Function. According to the presently
7 preferred embodiment, a lookup table 28, including a "patch center
8 function" (line 1 of the RAM table) provides the functionality
9 necessary to determine whether a code segment has a replacement in
10 RAM. Operation of the patch center function is described in more
11 detail below with reference to Figure 5 and Code Listing 2. The
12 lookup table is illustrated schematically at lines 2-4 of the RAM
13 table, line 2 being the start of table (SOT) and line 4 being the
14 end of table (EOT). Each entry in the lookup table refers to a
15 return address (RA) and a patch address (PA). For example, the
16 first entry refers to a return address of X+1 and a patch address
17 of 5. This signifies that the code being replaced is a
18 replacement (shown at 30 in Figure 4) for code segment X; the
19 replacement code begins at line 5 in the RAM table and upon
20 execution of the replacement code, execution continues at an
21 address specified in the replacement code, typically the next
22 segment after X. It is possible to replace only a portion of a
23 code segment and return to a line within the original code segment
24 to continue execution. As will be seen below in Code Listing 2,
25 it is necessary to replace a contiguous set of code lines from the

1 call function forward. Similar entries are indicated for code
2 segment Y and Z replacements which are illustrated at 32 and 34 in
3 Figure 4.

4
5 Referring now to Figure 5, when the patch center function is
6 called at 100, the lookup table entries are scanned at 102. If a
7 return address match is found at 104, program execution is
8 directed to the patch address at 106. The replacement code is
9 executed at 108, and program execution is returned to a ROM
10 address specified in the patch code at 110. So long as the end of
11 the table has not been reached as determined at 112, the patch
12 center function looks for a table entry for replacement code.
13 When it is determined that the end of the table has been reached,
14 program execution returns to the return address in ROM at 114. As
15 mentioned above, if there is no replacement code in RAM, program
16 execution returns to the return address specified by the call
17 function which also acts as an index to the lookup table. If
18 replacement code is executed, the return address is specified by
19 the replacement code.

20
21 The operation of the invention will be better understood with
22 reference to the following Code Listing 1 and Code Listing 2 which
23 illustrate the structure of the code in ROM and RAM respectively.

```

1 *****
2
3 * Code Listing 1 (ROM Segment)
4 *****
5
6 Code_in Rom:
7     Patch_Call1()           ; Function call
8     .
9     .
10    .
11    Patch_Call1()
12    .
13    .
14    .
15    Patch_Call2()
16 ROM_Return_Addr2:           ; Return Address (RA)
17     I = I + 10              ; Segment code starts
18     K + K - 55              ; Need modification !
19 ROM_Address2:
20     M = M + 88
21     .
22     .                       ; segment code ends
23     Patch_Call2()
24     J = J - 15
25     .
26     .
27     Patch_Call2()
28     .
29     .
30     .
31     Patch_Call3()
32     .
33     .
34     .
35     Patch_Call3()

```

Code Listing 1 (ROM Segment)

Referring to Code Listing 1, a code segment is shown with seven patch calls at lines 7, 11, 15, 23, 27, 31, and 35. Most of the code is not shown but is represented by dots, i.e. at lines 8-10, 12-14, 21, 22, 25, 26, 28-30, and 32-34. The code which will be modified follows patch call2() at line 15.

```

1  *****
2  * Code Listing 2 (RAM Segment)
3  *****
4  Code_in_RAM:
5
6  Patch_Call1                                ; Dummy function
7      Return                                ; Return to ROM Code
8
9  Patch_Call2:                                ; Dummy Function
10     Go to Patch_Center 2                    ; Replace Return here !
11
12 Patch_Call3:                                ; Dummy Function
13     Return                                ; Return to ROM Code
14     .
15     .
16     .
17
18 Patch_Center2:                              ; Patch Center Function
19
20     Save_Context()                          ; Save necessary registers
21
22     Address_Match = Patch_Address_Table_Search()
23                                           ; Lookup table searching
24
25     if (Address_Match == YES)
26         Go to Patch_Code2                    ; Have replacement code
27     else
28         Restore_Context()                    ; Restore the registers
29         Return                                ; Return to ROM code
30
31 Patch_Center2_Lookup_SOT:                    ; Start of Lookup Table
32     ROM_Return_Addr2                        ; Return Address (RA)
33     Patch_Code2                             ; Patch Address (PA)
34     .
35     .
36     .
37 Patch_Center_Lookup_EOT:                    ; End of Lookup Table
38
39 Patch_Code2:                                ; Patch Address (PA)
40     I = I + 10                               ; Replacement code
41     K = K - 99                               ; Replace K = K - 55 in ROM
42     Restore_Context()                        ; Restore the registers
43     Return to ROM_Address2
44
45

```

Code Listing 2 (RAM Segment)

48 Turning now to Code Listing 2, lines 6-13 represent the dummy
 49 functions. As shown, patch calls 1 and 3 are dummy functions
 50 which return the program to ROM. Patch call 2 directs the program

1 to patch center 2 which begins at line 18. Patch center 2 saves
2 the context and checks the address match. The address match
3 lookup table is illustrated at lines 31-37. If the address
4 matches, the program is directed to patch code 2 which starts at
5 line 39. If the address does not match, context is restored and
6 the program is returned to ROM. As shown at lines 39-43, patch
7 code 2 is designed to replace two lines of code in code listing 1,
8 i.e. replace lines 17 and 18 of code listing 1 with lines 34 and
9 35 of code listing 2. After executing lines 40 and 41 of code
10 listing 2, context is restored at line 42 and the program is
11 returned to ROM at 43. The return address is indicated in code
12 listings 1 and 2 as ROM_Address2. The actual return address is
13 derived from knowledge of the code in ROM. It should be noted
14 that line 40 of code listing 2 is identical to line 17 of code
15 listing 1. Nevertheless, it is replaced because, as mentioned
16 above, the function calls according to the invention allow and
17 require replacement of all code from the function call through the
18 corrected code.

19
20 There have been described and illustrated herein methods and
21 apparatus for modifying program code stored in ROM. While
22 particular embodiments of the invention have been described, it is
23 not intended that the invention be limited thereto, as it is
24 intended that the invention be as broad in scope as the art will
25 allow and that the specification be read likewise. It will

- 1 therefore be appreciated by those skilled in the art that yet
- 2 other modifications could be made to the provided invention
- 3 without deviating from its spirit and scope as so claimed.